



# Programación Python para BigData

**Lección 4: Base de datos: PostgreSQL**

# ÍNDICE

<b>Lección 4: Base de datos: PostgreSQL .....</b>	<b>2</b>
Presentación y objetivos.....	2
1. PostgreSQL.....	3
2. Adminer .....	13
3. Psycopg2.....	23
4. Puntos claves .....	32

## Lección 4: Base de datos: PostgreSQL

### PRESENTACIÓN Y OBJETIVOS

En esta lección aprenderemos a trabajar con un ejemplo de base de datos relacionable como es PostgreSQL, usando tres instrumentos para saber cómo leer, insertar, actualizar o eliminar un dato. (CRUD)



#### *Objetivos*

- | Trabajar a través de consola usando directamente PostgreSQL
- | Trabajar con Adminer es un visualizador de la base de datos.
- | Trabajar con una librería de Python como es Psycopg2

# 1. POSTGRESQL

## Introducción base de datos relacionales

La base de datos relacional (BDR) es un tipo de base de datos (BBDD) que cumple con el modelo relacional (el modelo más utilizado actualmente para implementar las BBDD ya planificadas). Todos los datos se almacenan y se accede a ellos por medio de relaciones previamente establecidas.

Las relaciones que almacenan datos son llamadas relaciones base y su implementación es llamada "tabla".

Otras relaciones no almacenan datos, pero son calculadas al aplicar operaciones relacionales. Estas relaciones son llamadas relaciones derivadas y su implementación es llamada "vista" o "consulta". Las relaciones derivadas son convenientes, ya que expresan información de varias relaciones actuando como si fuera una sola tabla.

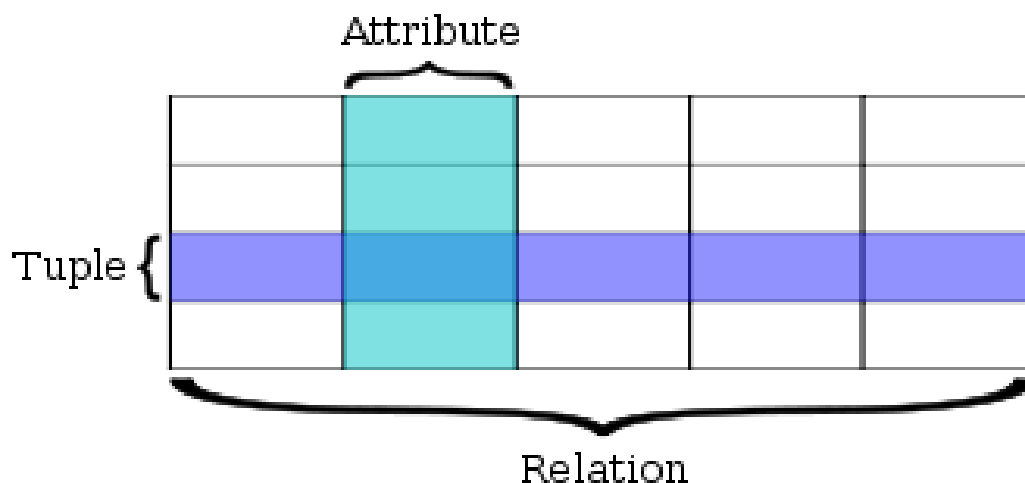


Figura 1.1 Estructura de la tabla/relación en SQL. Fuente Wikipedia

Término SQL	Término de bases de datos relacionales	Descripción
<i>Fila</i>	<i>Tupla o registro</i>	Un conjunto de datos, que representa un ítem simple
<i>Columna</i>	<i>Atributo o campo</i>	Un elemento etiquetado de una tupla, p.e. "Nombre" o "Fecha"
<i>Tabla</i>	Relación	Un conjunto de tuplas compartiendo los mismos atributos; un conjunto de filas y columnas.

## Clasificación de Claves

- | Clave primaria o Private key: Una clave primaria es una clave única (puede estar conformada por uno o más campos de la tabla) elegida entre todas las candidatas que define unívocamente a todos los demás atributos de la tabla para especificar los datos que serán relacionados con las demás tablas. La forma de hacer esto (relación entre tablas) es por medio de claves foráneas.
- | Clave externa o foránea o Foreign key: Una clave foránea es una referencia a una clave en otra tabla, determina la relación existente en dos tablas. Las claves foráneas no necesitan ser claves únicas en la tabla donde están y sí a donde están referenciadas.
- | Clave índice o ID: Las claves índice surgen con la necesidad de tener un acceso más rápido a los datos. Los índices pueden ser creados con cualquier combinación de campos de una tabla. Las consultas que filtran registros por medio de estos campos, pueden encontrar los registros de forma no secuencial usando la clave índice.

Las bases de datos relacionales incluyen múltiples técnicas de ordenamiento, cada una de ellas es óptima para cierta distribución de datos y tamaño de la relación.

Los índices generalmente no se consideran parte de la base de datos, pues son un detalle agregado. Sin embargo, las claves índices son desarrolladas por el mismo grupo de programadores que las otras partes de la base de datos.

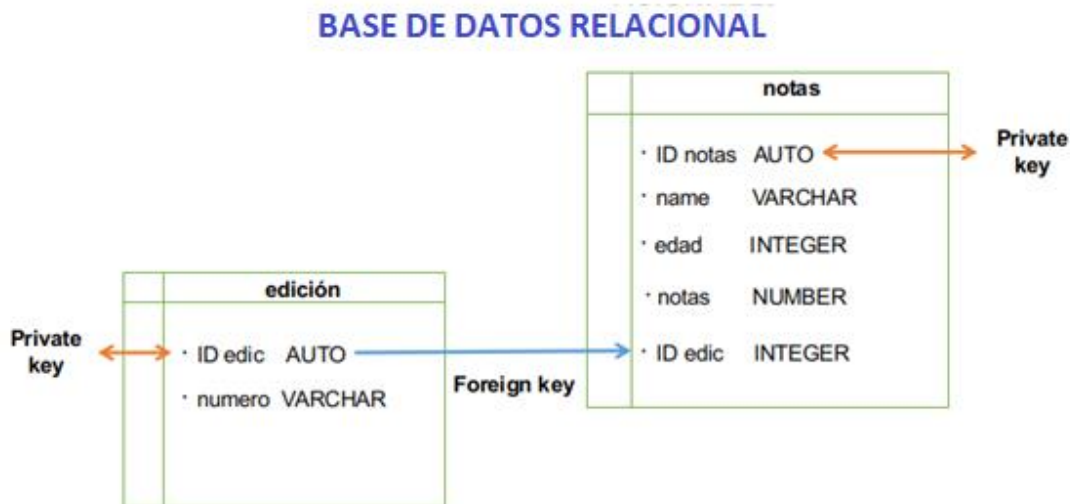


Figura 1.2 Estructura de relación entre tablas en SQL.

## Introducción a PostgreSQL

PostgreSQL, o conocido como Postgres es un sistema de código abierto de administración de bases de datos del **tipo relacional o SQL**, aunque también es posible ejecutar consultas que sean no relaciones. En este sistema, las consultas relacionales se basan en SQL, mientras que las no relacionales hacen uso de JSON.

Es gratuito, y su desarrollo es llevado adelante por una gran comunidad de colaboradores de todo el mundo para hacer de este sistema una de las opciones más sólidas a nivel de bases de datos.

Dos detalles a destacar de PostgreSQL es que posee data types (tipos de datos) avanzados y permite ejecutar optimizaciones de rendimiento avanzadas, que son características que por lo general solo se ven en sistemas de bases de datos comerciales, como por ejemplo SQL Server de Microsoft u Oracle de la compañía homónima.

## Características

- | Es de código abierto.
- | Es gratuito.
- | Es multiplataforma: es un software que puede correr bajo distintos entornos y sistemas operativos.
- | Es fácil de usar
- | Puede manejar un gran volumen de datos.
- | Soporte total de ACID: siglas de Atomicity, Consistency, Isolation y Durability, que si lo traducimos al español básicamente hablan de la atomicidad, consistencia, aislamiento y durabilidad de las transacciones que se realizan en una base de datos. Esto es importante ya que da la seguridad de que, si se produce una falla durante una transacción, los datos no se perderán ni terminarán donde no deban.

## Comparación de PostgreSQL vs MySQL

PostgreSQL es un sistema de código libre y totalmente gratuito que está en manos de una comunidad. Con MySQL sigue existiendo una versión gratuita aunque eso solo será hasta que Oracle así lo quiera, ya que hace unos años pertenece a ellos. Afortunadamente existen alternativas MySQL extremadamente similares como MariaDB, por si algún día necesitas un reemplazo.

En lo que refiere a las consultas o queries, las soportadas por MySQL tienden a ser más simples, mientras que PostgreSQL soporta queries más complejas, lo cual le ha valido el título del sistema de bases de datos relacionales más avanzado del mercado.

Rendimientos:

- | MySQL es mejor si se manejan bajos volúmenes de datos, es decir, si tenemos bases de datos pequeñas o medianas a las cuales se hagan una cantidad de consultas no muy alta.
- | PostgreSQL es mejor a la hora de manejar volúmenes de datos grandes y se suele usar más cuando se tienen bases de datos grandes y con alta cantidad de consultas.

Cumplimiento de ACID:

- | MySQL no lo soporta al 100%, sino solamente las tablas que hagan uso de NDB Cluster o InnoDB.
- | PostgreSQL sí lo soporta completo en todos los casos.

Por todo ello como nuestra asignatura es Big Data usaremos como base de datos PostgreSQL

## Ejecutar PostgreSQL

Realizaremos las actividades descritas como **CRUD**, que viene del Inglés (Create, Read, Update and Delete):

**Create / Crear:** crearemos un nuevo dato en nuestra tabla.

**Read / leer:** leeremos los datos de la tabla.

**Update / actualizar:** actualizaremos un dato ya existente en la tabla.

**Delete / eliminar:** eliminamos un dato de la tabla.

Para ello usamos el docker-compose creado en la actividad de la lección 2, donde tendremos activos los servicio de PostgreSQL y adminer para la visualización y gestión de los datos.

Necesitamos para trabajar con comandos en el PostgreSQL ejecutar la imagen:



```
isabel@isabel-SVE1512E1EW:~/atom$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
473837faeb01   adminer        "entrypoint.sh docke..." 8 minutes ago  Up 36 seconds  0.0.0.0:8080->8080/tcp, :::8080->8080/tcp  atom_adminer_1
9ce68f58dbe4   postgres      "docker-entrypoint.s..." 8 minutes ago  Up 36 seconds  0.0.0.0:5432->5432/tcp, :::5432->5432/tcp  atom_db_1
```

Figura 1.3 Imágenes ejecutadas a través de docker-compose.

```
sudo docker exec -it ato_db_1 bash

psql -U IsaMan
```

-U es el usuario descrito en la imagen al ejecutar el docker-compose

```
isabel@isabel-SVE1512E1EW:~/atom$ sudo docker exec -it atom_db_1 bash
root@9ce68f58dbe4:/# psql -U IsaMan
psql (13.3 (Debian 13.3-1.pgdg100+1))
Type "help" for help.

IsaMan=# ^C
IsaMan=#
```

Figura 1.4 Ejecución de PostgreSQL

## Creamos una base de datos

```
CREATE DATABASE <nombre de la base de datos>
```

```
IsaMan=# CREATE DATABASE test;
CREATE DATABASE
```

Figura 1.5 Crear un base de datos en PostgreSQL

## Eliminar una base de datos

```
DROP DATABASE <nombre de la base de datos>
```

```
IsaMan=# DROP DATABASE test;
DROP DATABASE
IsaMan=#
```

Figura 1.6 Eliminar base de datos en PostgreSQL

## Crear una tabla en la base de datos

```
CREATE TABLE <nombre_tabla>(<nombre_columna> <tipo_variable>,...);
```

```
IsaMan=# CREATE TABLE notas(name varchar(80), age int, grades decimal, date date);
CREATE TABLE
```

Figura 1.7 Crear una tabla en la base de datos en PostgreSQL

Tipos de Variables		
Número	bigint (tamaño)	Número de gran tamaño entero
	int (tamaño)	Número entero
	smallint(tamaño)	Número entero pequeño tamaño
	real	También decimal. Datos numéricos de precisión flotante
	money	Datos monetarios
	boolean	Booleano significa si/no, on/off, etc
	numeric	Numérico
	double precision	Decimal con precisión
Fecha y hora	date	Una fecha. Formato: AAAA-MM-DD
	time	Un tiempo. Formato: HH: MI: SS
	timestamp	Marca de tiempo en segundos
	timestampz	Marca de tiempo en segundos UTC
	interval	Intervalo de tiempo

Cadena	Character(tamaño)	o también llamado char.cadena de caracteres
	Character varying(tamaño)	o también llamado varchar. cadena de caracteres de ancho variable
	text	Cadena de caracteres de ancho variable
	ts query	Almacena en formato cursor
	ts vector	Almacena en formato vector
	uuid	Almacenar en formato UUID
	json	Almacenar en formato JSON
	xml	Almacenar en formato XML
	jsonb	Almacenar en formato BSON
Binario	bit	Entero que puede ser 0, 1 o NULL
	bit varying	Entero que puede ser variable
	bytea	cadena de caracteres binaria
Red	cidr	
	inet	
	macaddr	
	txid_snapshot	
Geometría	box	Formato caja
	circle	Formato de círculo
	line	Formato de línea
	lseg	
	path	Formato de ruta
	point	Formato de puntos
	polygon	Formato de Polígono

## Insertar dato

```
INSERT INTO <nombre tabla> VALUES (<valores>)
```

INSERT INTO notas VALUES ('Isabel Maniega', 205, 5.4, '07/07/2021')

```
IsaMan=# INSERT INTO notas VALUES ('Isabel Maniega', 205, 5.4, '07/07/2021');
INSERT 0 1
IsaMan=#
```

Figura 1.8 Insertar un datos en la tabla en la base de datos en PostgreSQL



NOTA: La fecha en formato americano (MES/ DIA/ AÑO)

## Leer datos

```
SELECT * FROM <nombre_tabla>;
```

\* Significa seleccionar todas las columnas de la tabla.

También se puede poner en vez de \* el nombre de la columna/s

```
SELECT * FROM notas;
```

```
IsaMan=# SELECT * FROM notas;
      name      | age | grades |      date
-----+-----+-----+-----
 Isabel Maniega | 205 |    5.4 | 2021-07-07
(1 row)
```

Figura 1.9 Seleccionar datos en la tabla en la base de datos en PostgreSQL

## Actualizar datos

```
UPDATE <nombre_tablas> SET columna=valor_a_modificar;
```

```
UPDATE notas SET grades=6.9;
```

```
IsaMan=# UPDATE notas SET grades=6.9;
UPDATE 1
IsaMan=#
```

Figura 1.10 Actualizar datos en la tabla en la base de datos en PostgreSQL

## Eliminar datos

```
DELETE FROM <nombre_tabla> WHERE columna=valor;
```

```
DELETE FROM notas WHERE name='Isabel Maniega'
```

```
IsaMan=# DELETE FROM notas WHERE name='Isabel Maniega';  
DELETE 1  
IsaMan=#
```

*Figura 1.11 Eliminar datos en la tabla en la base de datos en PostgreSQL*

## Eliminar tabla

```
DROP TABLE <nombre_tabla>;
```

```
IsaMan=# DROP TABLE notas;  
DROP TABLE
```

*Figura 1.12 Eliminar tabla en la base de datos en PostgreSQL*

## 2. ADMINER

Adminer es una aplicación para gestionar base de datos SQL, como hemos usado docker-compose según la descripción de la imagen PostgreSQL, viene el servicio Adminer que usaremos para gestionar la base de datos usando una aplicación.

Para ello sólo necesitamos ir al navegador y poner:

<http://localhost:8080/>

← → ↻ ⓘ localhost:8080/?pgsql=db

Aplicaciones Puesta en mar... Tutorial de Dj... Get Started Wi...

Idioma: Español

Adminer 4.8.1 Login

Sesión finalizada con éxito. Thanks for using Adminer, consider donating.

Motor de base de datos	PostgreSQL
Servidor	db
Usuario	saMan
Contraseña	*****
Base de datos	

Login ☐ Guardar contraseña

Figura 2.1 Pantalla inicial de la aplicación Adminer

**Motor de base de datos:** PostgreSQL

**Servidor:** db

**Usuario:** el puesto en docker-compose

**Contraseña:** el puesto en docker compose

**Base de datos:** vacio

Pulsamos Login



Figura 2.2 Pantalla donde podemos ver nuestras bases de datos activas en PostgreSQL

## Creamos una base de datos

Seleccionamos crear Base de datos:



Figura 2.3 Crear bases de datos en PostgreSQL

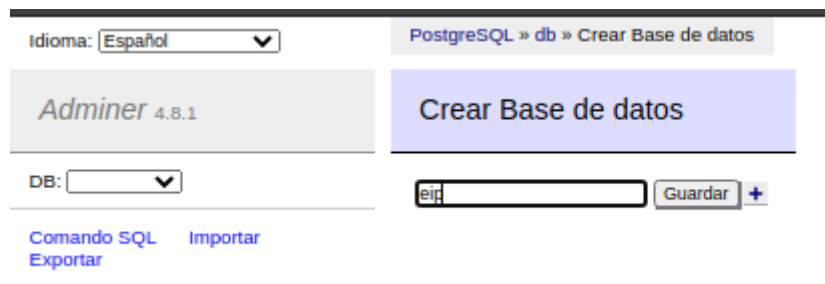


Figura 2.4 Nombrar la base de datos en PostgreSQL

Ponemos “eip” y pulsamos Guardar.

## Crear una tabla en la base de datos

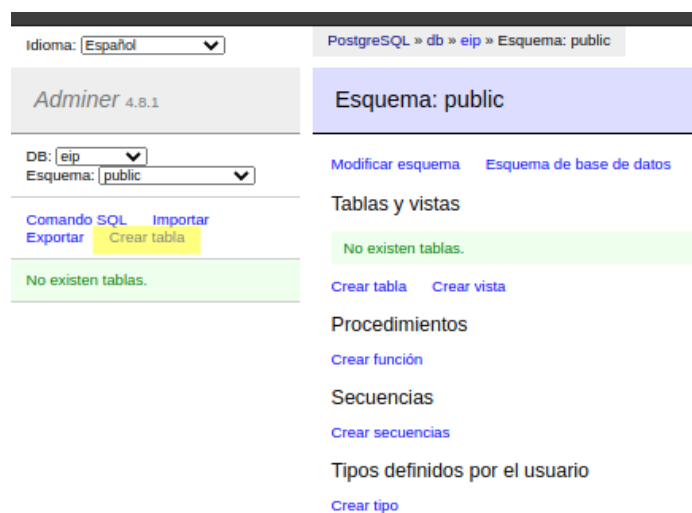


Figura 2.5 Base de datos “eip” en PostgreSQL

Pulsar crear tabla:

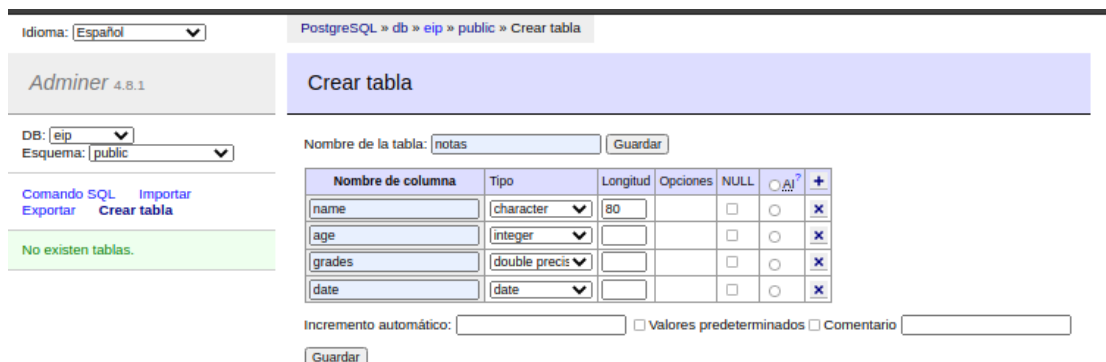


Figura 2.6 Tabla notas en base de datos “eip” en PostgreSQL



Pondremos:

**Nombre de la tabla:** notas

**Name:** character 80

**Age:** integer

**Grades:** double precision

**Date:** date

Pulsaremos Guardar:

Idioma: Español

PostgreSQL » db » eip » public » Tabla: notas

**Adminer 4.8.1**

DB: eip  
Esquema: public

[Comando SQL](#) [Importar](#)  
[Exportar](#) [Crear tabla](#)

[registros notas](#)

**Tabla: notas**

Tabla creada. 09:59:14 [Comando SQL](#)

[Visualizar contenido](#) [Mostrar estructura](#) [Modificar tabla](#) [Nuevo Registro](#)

Columna	Tipo	Comentario
name	character(80)	
age	integer	
grades	double precision	
date	date	

Índices  
[Modificar índices](#)

Claves externas  
[Agregar clave externa](#)

Disparadores  
[Agregar disparador](#)

Figura 2.7 Tabla notas en base de datos "eip" en PostgreSQL

Ya nos ha creado la nueva tabla.

## Insertar datos

Pulsamos nuevo registro:



Idioma: Español

PostgreSQL » db » eip » public » Tabla: notas

**Adminer 4.8.1**

DB: eip  
Esquema: public

[Comando SQL](#) [Importar](#)  
[Exportar](#) [Crear tabla](#)

[registros notas](#)

**Tabla: notas**

Tabla creada. 08:28:14 [Comando SQL](#)

[Visualizar contenido](#) [Mostrar estructura](#) [Modificar tabla](#) [Nuevo Registro](#)

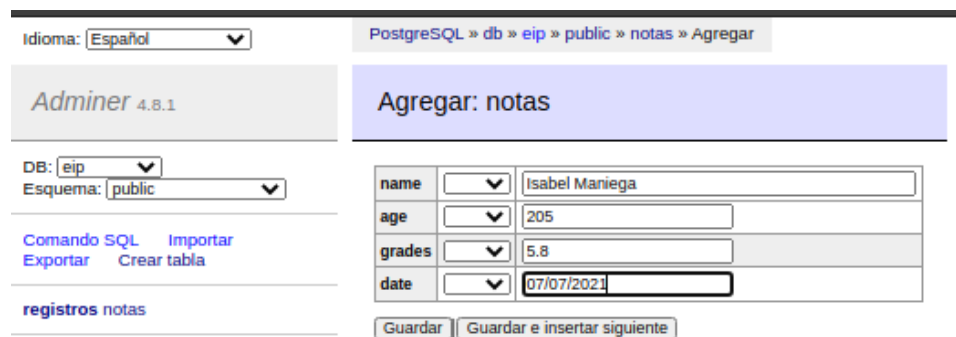
Columna	Tipo	Comentario
name	character(80)	
age	integer	
grades	double precision	
date	date	

Índices  
[Modificar índices](#)

Claves externas  
[Agregar clave externa](#)

Disparadores  
[Agregar disparador](#)

Figura 2.8 Insertar datos en tabla notas en base de datos "eip" en PostgreSQL



Idioma: Español

PostgreSQL » db » eip » public » notas » Agregar

**Adminer 4.8.1**

DB: eip  
Esquema: public

[Comando SQL](#) [Importar](#)  
[Exportar](#) [Crear tabla](#)

[registros notas](#)

**Agregar: notas**

name	<input type="text" value="Isabel Maniega"/>
age	<input type="text" value="205"/>
grades	<input type="text" value="5.8"/>
date	<input type="text" value="07/07/2021"/>

Figura 2.9 Insertar datos en tabla notas en base de datos "eip" en PostgreSQL

Insertamos los datos como la imagen y pulsamos guardar:



Figura 2.10 Insertar datos en tabla notas en base de datos "eip" en PostgreSQL

Podemos ver los datos que hemos introducido.

## Actualizar un dato



Figura 2.11 Modificar datos en tabla notas en base de datos "eip" en PostgreSQL

Pulsaremos modify:

Idioma: Español

PostgreSQL » db » eip » public » Mostrar: notas

**Adminer 4.8.1**

DB: eip  
Esquema: public

[Comando SQL](#) [Importar](#)  
[Exportar](#) [Crear tabla](#)

[registros notas](#)

**Mostrar: notas**

[Visualizar contenido](#) [Mostrar estructura](#) [Modificar tabla](#) [Nuevo Registro](#)

[Mostrar](#) [Condición](#) [Ordenar](#) [Limite](#) [Longitud de texto](#) [Acción](#)

[50](#) [100](#) [Mostrar](#)

**SELECT \* FROM "notas" LIMIT 50 (0.001 s) [Modificar](#)**

<input type="checkbox"/> Modify	name	age	grades	date
<input type="checkbox"/> modificar	Isabel Maniega	205	<input type="text" value="7.6"/>	2021-07-07
<input type="checkbox"/> modificar	José Manuel Peña	210	6.8	2021-07-07

[Resultado completo](#) [Modify](#) [Selected \(0\)](#) [Exportar \(2\)](#)

☐ 2 registros [Guardar](#) [Modificar](#) [Clonar](#) [Eliminar](#)

[Importar](#)

Figura 2.12 Modificar datos en tabla notas en base de datos "eip" en PostgreSQL

Modificamos la nota y pulsamos

Probamos con comandos:

Idioma: Español

PostgreSQL » db » eip » public » Comando SQL

**Adminer 4.8.1**

DB: eip  
Esquema: public

[Comando SQL](#) [Importar](#)  
[Exportar](#) [Crear tabla](#)

[registros notas](#)

**Comando SQL**

`UPDATE notas SET grades=7.9 WHERE name='Isabel Maniega';`

[Ejecutar](#) [Limit rows:](#)  ☐ Parar en caso de error ☐ Mostrar solamente errores

[Histórico](#)

Figura 2.13 Modificar datos en tabla notas en base de datos "eip" en PostgreSQL

```
UPDATE notas SET grades=7.9 WHERE name='Isabel Maniega';
```

Ejecutamos:

Idioma: Español PostgreSQL » db » eip » public » Comando SQL

**Adminer 4.8.1**

DB: eip  
Esquema: public

**Comando SQL** [Importar](#)  
[Exportar](#) [Crear tabla](#)

[registros notas](#)

**Comando SQL**

`UPDATE notas SET grades=7.9 WHERE name='Isabel Maniega'`

Consulta ejecutada, 1 registro afectado. (0.002 s) [Modificar](#)

`UPDATE notas SET grades=7.9 WHERE name='Isabel Maniega';`

[Ejecutar](#) Limit rows:  ☐ Parar en caso de error ☐ Mostrar solamente errores

[Histórico](#)

Figura 2.14 Modificar datos en tabla notas en base de datos "eip" en PostgreSQL

Pulsamos Modificar:

Si volvemos a la página de notas a Visualizar contenido vemos que nos lo ha modificado:

Idioma: Español PostgreSQL » db » eip » public » Mostrar: notas

**Adminer 4.8.1**

DB: eip  
Esquema: public

**Mostrar: notas**

[Visualizar contenido](#) [Mostrar estructura](#) [Modificar tabla](#) [Nuevo Registro](#)

[Mostrar](#) [Condición](#) [Ordenar](#) [Limite](#) [Longitud de texto](#) [Acción](#)

[Mostrar](#)

`SELECT * FROM "notas" LIMIT 50 (0.001 s) Modificar`

	name	age	grades	date
<a href="#">Modificar</a>	José Manuel Peña	210	6.8	2021-07-07
<a href="#">Modificar</a>	Isabel Maniega	205	7.9	2021-07-07

Resultado completo ☐ 2 registros [Modificar](#) [Guardar](#) [Modificar](#) [Clonar](#) [Eliminar](#) [Exportar \(2\)](#)

[Importar](#)

Figura 2.15 Modificar datos en tabla notas en base de datos "eip" en PostgreSQL

## Eliminar un dato

Seleccionamos el dato y pulsamos eliminar nos aparece un aviso si estamos seguros, damos aceptar:

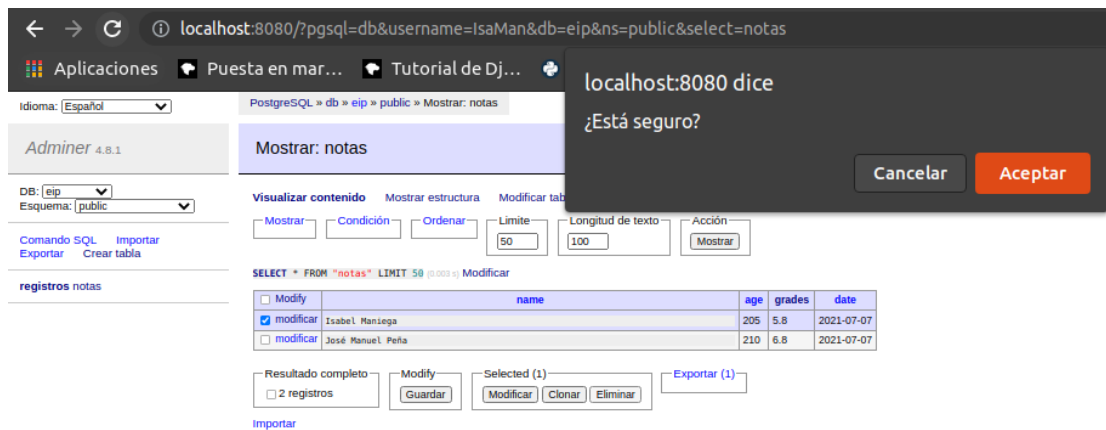


Figura 2.16 Eliminar datos en tabla notas en base de datos "eip" en PostgreSQL


Por comandos sería:



Figura 2.17 Eliminar datos en tabla notas en base de datos "eip" en PostgreSQL

```
DELETE FROM notas WHERE name='Isabel Maniega';
```

Pulsamos ejecutar



Idioma: Español

PostgreSQL » db » eip » public » Comando SQL

**Comando SQL**

DELETE FROM notas WHERE name='Isabel Maniega';

Consulta ejecutada, 1 registro afectado. (0.003 s) [Modificar](#)

DELETE FROM notas WHERE name='Isabel Maniega';

[Ejecutar](#) Limit rows:  ☐ Parar en caso de error ☐ Mostrar solamente errores

[Histórico](#)

Figura 2.18 Eliminar datos en tabla notas en base de datos "eip" en PostgreSQL

Pulsaremos después a Modificar.

Volvemos a la ventana principal a Visualizar contenido:



Idioma: Español

PostgreSQL » db » eip » public » Mostrar: notas

**Mostrar: notas**

[Visualizar contenido](#) [Mostrar estructura](#) [Modificar tabla](#) [Nuevo Registro](#)

[Mostrar](#) [Condición](#) [Ordenar](#) [Limite](#) [Longitud de texto](#) [Acción](#)

[SELECT \\* FROM "notas" LIMIT 50 \(0.001 s\)](#) [Modificar](#)

	name	age	grades	date
<input type="checkbox"/> modificar	José Manuel Peña	210	6.8	2021-07-07

☐ Resultado completo ☐ 1 registro ☐ Modify     [Exportar \(1\)](#)

[Importar](#)

Figura 2.19 Eliminar datos en tabla notas en base de datos "eip" en PostgreSQL

### 3. PSYCOPG2

Librería de Python para gestión de base de datos en PostgreSQL.

Para ello necesitamos crear el entorno virtual:

```
isabel@isabel-SVE1512E1EW:~/atom/postgres$ virtualenv env
created virtual environment CPython3.8.10.final.0-64 in 294ms
creator CPython3Postix(dest=/home/isabel/atom/postgres/env, clear=False, global=False)
seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest, pkg_resources=latest, via=copy, app_data_dir=/home/isabel/.local/share/virtualenv/seed-app-data/v1.0.1.debian.1)
activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator
isabel@isabel-SVE1512E1EW:~/atom/postgres$
```

Figura 3.1 Creación de entorno virtual

En atom creamos el archivo "postgres.py":

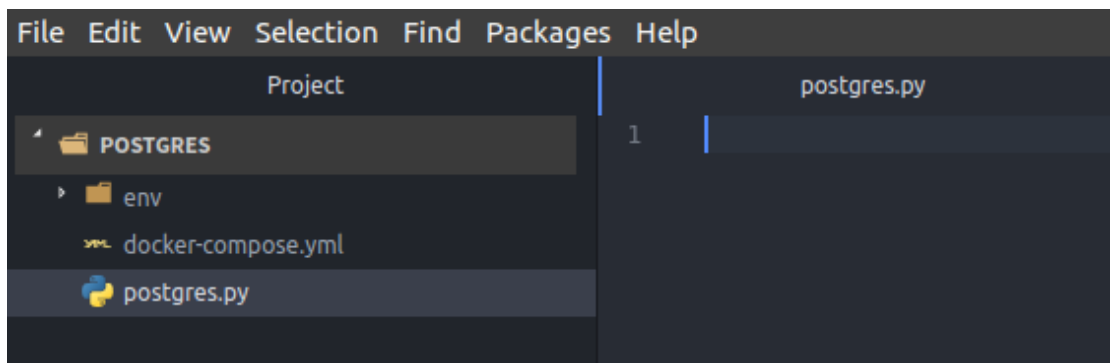


Figura 3.2 Creación de archivo Python

Activamos el entorno e instalamos la librería:

```
pip install psycopg2-binary
```

```
(env) isabel@isabel-SVE1512E1EW:~/atom/postgres$ pip install psycopg2-binary
Collecting psycopg2-binary
  Downloading psycopg2-binary-2.9.1-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.4 MB)
    | 3.4 MB 2.6 MB/s
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.9.1
```

Figura 3.3 Creación de archivo Python

Para conectarnos a la base de datos ponemos:

```
import psycopg2
# Connect to an existing database
conn = psycopg2.connect(user='IsaMan', password='cursoPython',
                        host='localhost', port=5432)

# Open a cursor to perform database operations
cur = conn.cursor()
```



```
postgres.py
1  import psycopg2
2
3
4  # Connect to an existing database
5  conn = psycopg2.connect(user='IsaMan', password='cursoPython',
6                          host='localhost', port=5432)
7
8
9  # Open a cursor to perform database operations
10 cur = conn.cursor()
```

Figura 3.4 Conexión a la base de datos

## Crear una base de datos nueva

Añadimos las siguientes librerías:

```
from psycopg2 import sql
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
```

Colocamos entre la conexión y el cursor:

```
conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
```

Después creamos la base de datos:

```
# sql.SQL and sql.Identifier are needed to avoid SQL
# injection attacks.
cur.execute(sql.SQL('CREATE DATABASE {};').format(
    sql.Identifier('eip')))
# Close communication with the database
cur.close()
conn.close()
```

```

2  from psycopg2 import sql
3  from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
4
5  # Connect to an existing database
6  conn = psycopg2.connect(user='IsaMan', password='cursoPython',
7                          host='localhost', port=5432)
8
9  conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
10
11 # Open a cursor to perform database operations
12 cur = conn.cursor()
13
14 # sql.SQL and sql.Identifier are needed to avoid SQL injection attacks.
15 try:
16     cur.execute(sql.SQL('CREATE DATABASE {}').format(
17                 sql.Identifier('eip')))
18 except psycopg2.Error as e:
19     print('Error Crear base de datos: %s', str(e))
20
21 # Close communication with the database
22 cur.close()
23 conn.close()

```

Figura 3.5 Creación de base de datos llamada "eip"

```

(env) isabel@isabel-SVE1512E1EW:~/atom/postgres$ python postgres.py
(env) isabel@isabel-SVE1512E1EW:~/atom/postgres$

```

Figura 3.6 Ejecutar el script

Idioma: Español PostgreSQL » db

Adminer 4.8.1

DB:

Comando SQL Importar Exportar

Seleccionar Base de datos

[Crear Base de datos](#) [Lista de procesos](#) [Variables](#)

Versión PostgreSQL: **13.3 (Debian 13.3-1.pgdg100+1)** a través de la extensión de PHP PDO\_PgSQL

Logueado como: IsaMan

Base de datos - Refrescar	Colación	Tablas	Size - Compute
<input type="checkbox"/> IsaMan	en_US.utf8	?	?
<input type="checkbox"/> eip	en_US.utf8	?	?
<input type="checkbox"/> postgres	en_US.utf8	?	?
<input type="checkbox"/> template0	en_US.utf8	?	?
<input type="checkbox"/> template1	en_US.utf8	?	?

Selected (0)

Eliminar

Figura 3.7 Visualización de la base de datos "eip" en Adminer

Observamos que se ha creado.

## Crear una tabla

Una vez creada creamos una tabla nueva, debemos antes de seguir comentar la parte de crear la base de datos, para no tener problemas con el código.

Añadimos en conn la variable database donde vamos a conectar y la parte para crear la tabla:

```
# Connect to an existing database
conn = psycopg2.connect(database="eip", user='IsaMan',
                        password='cursoPython',
                        host='localhost', port=5432)

# Creamos una nueva tabla:
try:
    cur.execute("CREATE TABLE notas (name varchar(80), age
                int, grades real, date date);")
except psycopg2.Error as e:
    print('Error Crear tabla: %s', str(e))
```

Añadimos al final:

```
conn.commit()
```

Para hacer que los cambios sean persistentes

```
1  import psycopg2
2
3  # Connect to an existing database
4  conn = psycopg2.connect(database="eip", user='IsaMan', password='cursoPython',
5                          host='localhost', port=5432)
6
7
8  # Open a cursor to perform database operations
9  cur = conn.cursor()
10
11 # Creamos una nueva tabla:
12 try:
13     cur.execute("CREATE TABLE notas (name varchar(80), age int, grades real, date date);")
14 except psycopg2.Error as e:
15     print('Error Crear tabla: %s', str(e))
16
17
18 # Close communication with the database
19 conn.commit()
20 cur.close()
21 conn.close()
22
```

Figura 3.8 Creación de la tabla en base de datos "eip"



Figura 3.9 Visualización de la tabla notas en la base de datos "eip" en Adminer

## Insertar un dato

Una vez creada la tabla, debemos antes de seguir comentar la parte de crear la tabla, para no tener problemas con el código.

Pondremos:

```
# Pass data to fill a query placeholders and let Psycopg
perform
# the correct conversion (no more SQL injections!)
try:
    cur.execute("INSERT INTO notas VALUES
(%s, %s, %s, %s)",
                ('Isabel Maniega', 205, 5.6, '07/07/2021'))
except psycopg2.Error as e:
    print('Error Insertar dato: %s', str(e))
```

```

1  import psycopg2
2
3  # Connect to an existing database
4  conn = psycopg2.connect(database="eip", user='IsaMan', password='cursoPython',
5                          host='localhost', port=5432)
6
7  # Open a cursor to perform database operations
8  cur = conn.cursor()
9
10
11 # Pass data to fill a query placeholders and let Psycopg perform
12 # the correct conversion (no more SQL injections!)
13 try:
14     cur.execute("INSERT INTO notas VALUES (%s, %s, %s, %s)",
15                 ('Isabel Maniega', 205, 5.6, '07/07/2021'))
16 except psycopg2.Error as e:
17     print('Error Insertar dato: %s', str(e))
18 #
19
20
21 # Make the changes to the database persistent
22 conn.commit()
23
24 # Close communication with the database
25 cur.close()
26 conn.close()

```

Figura 3.10 Insertar un dato en la tabla notas en la base de datos "eip"

Idioma: Español

PostgreSQL » db » eip » public » Mostrar: notas

**Adminer 4.8.1**

DB: eip  
Esquema: public

Comando SQL [Importar](#)  
[Exportar](#) [Crear tabla](#)

registros notas

**Mostrar: notas**

Visualizar contenido   Mostrar estructura   Modificar tabla   Nuevo Registro

[Mostrar](#)   [Condición](#)   [Ordenar](#)   Limite:    Longitud de texto:    Acción: [Mostrar](#)

SELECT \* FROM "notas" LIMIT 50 (0.001 s) [Modificar](#)

<input type="checkbox"/> Modify	name	age	grades	date
<input type="checkbox"/> modificar	Isabel Maniega	205	5.6	2021-07-07

Resultado completo: ☐ 1 registro   Modify: [Guardar](#)   Selected (0): [Modificar](#) [Clonar](#) [Eliminar](#)   [Exportar \(1\)](#)

[Importar](#)

Figura 3.11 Visualización de la tabla notas en la base de datos "eip" en Adminer

## Lectura de los datos

```
# Query the database and obtain data as Python objects
cur.execute("SELECT * FROM notas;")
rows = cur.fetchall()
for row in rows:
    print(row)
```

```
1  import psycopg2
2
3  # Connect to an existing database
4  conn = psycopg2.connect(database="eip", user='IsaMan', password='cursoPython',
5                          host='localhost', port=5432)
6
7  # Open a cursor to perform database operations
8  cur = conn.cursor()
9
10 # Query the database and obtain data as Python objects
11 cur.execute("SELECT * FROM notas;")
12 rows = cur.fetchall()
13 for row in rows:
14     print(row)
15 #
16 # # Make the changes to the database persistent
17 conn.commit()
18
19 # Close communication with the database
20 cur.close()
21 conn.close()
22
```

Figura 3.12 Lectura datos en la tabla notas en la base de datos "eip"

## Actualizar un dato

```
# actualizar datos buscando el dato a actualizar
try:
    cur.execute("UPDATE notas SET grades=7.9 WHERE
name='Isabel Maniega'")
except psycopg2.Error as e:
    print('Error Actualizar dato: %s', str(e))
```

```

1 import psycopg2
2
3 # Connect to an existing database
4 conn = psycopg2.connect(database="eip", user='IsaMan', password='cursoPython',
5                           host='localhost', port=5432)
6
7 # Open a cursor to perform database operations
8 cur = conn.cursor()
9
10 # actualizar datos buscando el dato a actualizar
11 try:
12     cur.execute("UPDATE notas SET grades=7.9 WHERE name='Isabel Maniega'")
13 except psycopg2.Error as e:
14     print('Error Insertar dato: %s', str(e))
15
16 # Query the database and obtain data as Python objects
17 cur.execute("SELECT * FROM notas;")
18 rows = cur.fetchall()
19 for row in rows:
20     print(row)
21 #
22 # # Make the changes to the database persistent
23 conn.commit()
24
25 # Close communication with the database
26 cur.close()
27 conn.close()

```

Figura 3.13 Actualizar datos en la tabla notas en la base de datos "eip"

```

(env) isabel@isabel-SVE1512E1EW:~/atom/postgres$ python postgres.py
('Isabel Maniega', 205, 7.9, datetime.date(2021, 7, 7))

```

Figura 3.14 Salida consola: actualizar datos en la tabla notas en la base de datos "eip"

## Eliminar los datos

```

try:
    cur.execute("DELETE FROM notas WHERE name='Isabel
Maniega';")
except psycopg2.Error as e:
    print('Error eliminar dato: %s', str(e))

```

```

1  import psycopg2
2
3  # Connect to an existing database
4  conn = psycopg2.connect(database="eip", user='IsaMan', password='cursoPython',
5                          host='localhost', port=5432)
6
7  # Open a cursor to perform database operations
8  cur = conn.cursor()
9
10 try:
11     cur.execute("DELETE FROM notas WHERE name='Isabel Maniega';")
12 except psycopg2.Error as e:
13     print('Error eliminar dato: %s', str(e))
14
15 # Query the database and obtain data as Python objects
16 cur.execute("SELECT * FROM notas;")
17 rows = cur.fetchall()
18 for row in rows:
19     print(row)
20 #
21 # # Make the changes to the database persistent
22 conn.commit()
23
24 # Close communication with the database
25 cur.close()
26 conn.close()

```

Figura 3.15 Eliminar datos en la tabla notas en la base de datos "eip"

```

(env) isabel@isabel-SVE1512E1EW:~/atom/postgres$ python postgres.py
('José Manuel Peña', 215, 8.6, datetime.date(2021, 7, 7))

```

Figura 3.16 Salida consola: Eliminar datos en la tabla notas en la base de datos "eip"



## 4. PUNTOS CLAVES



### *No te olvides...*

- | PostgreSQL es un base de datos de tipo relacionales o SQL.
- | Para visualizar los datos de manera fácil usaremos Adminer.
- | Psycopg2 es una librería de Python que nos facilita la gestión de la base de datos.

